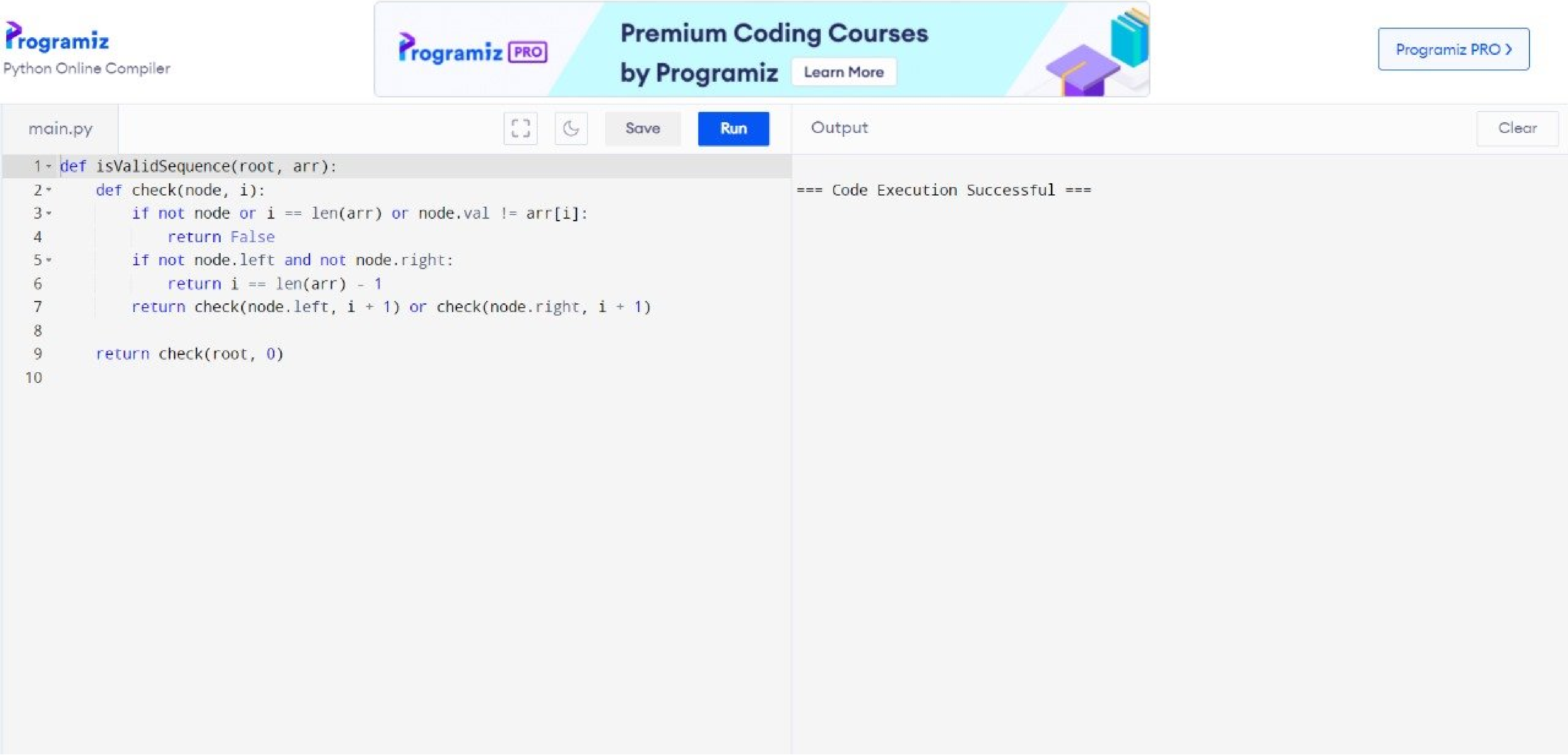
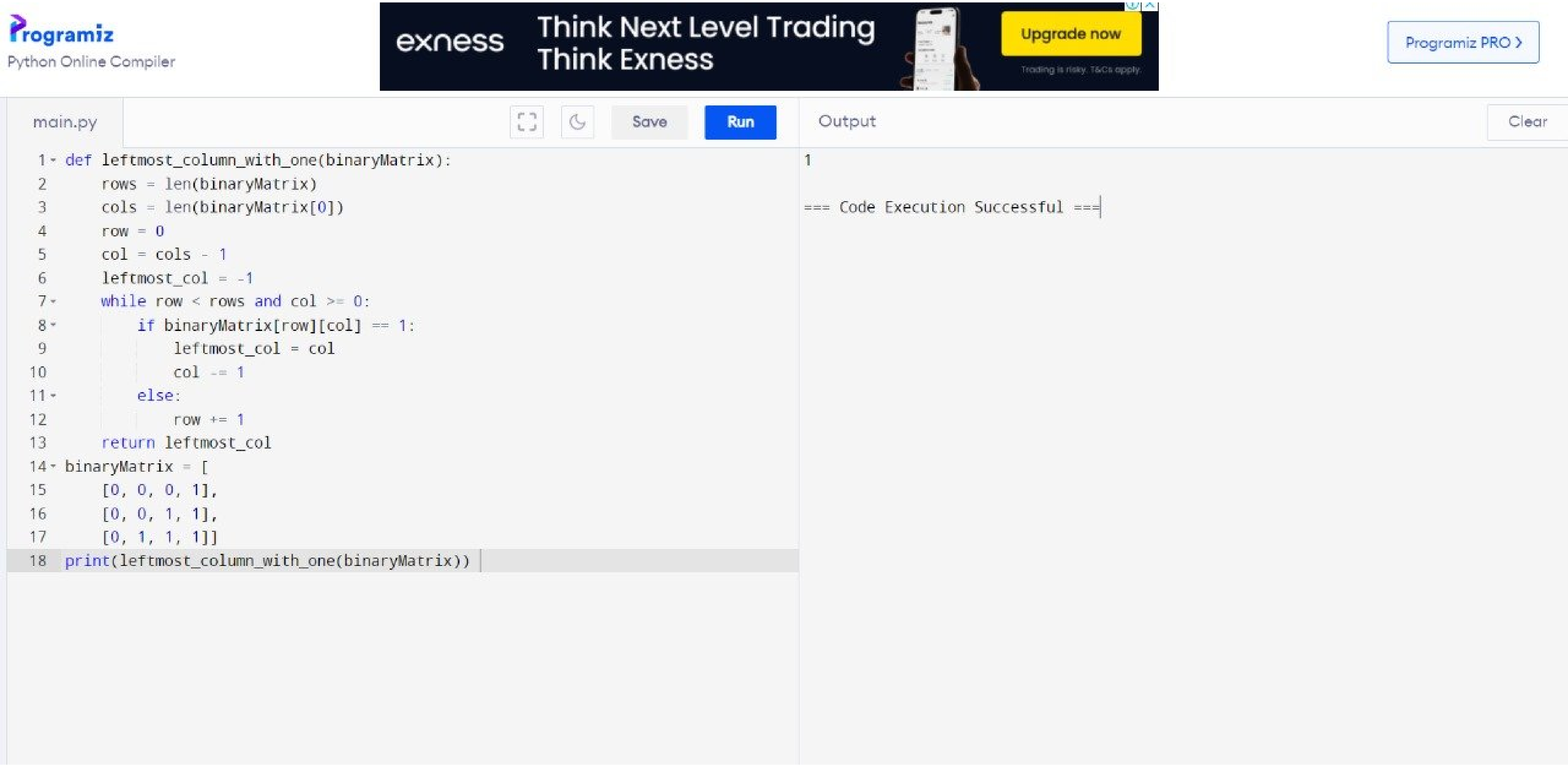


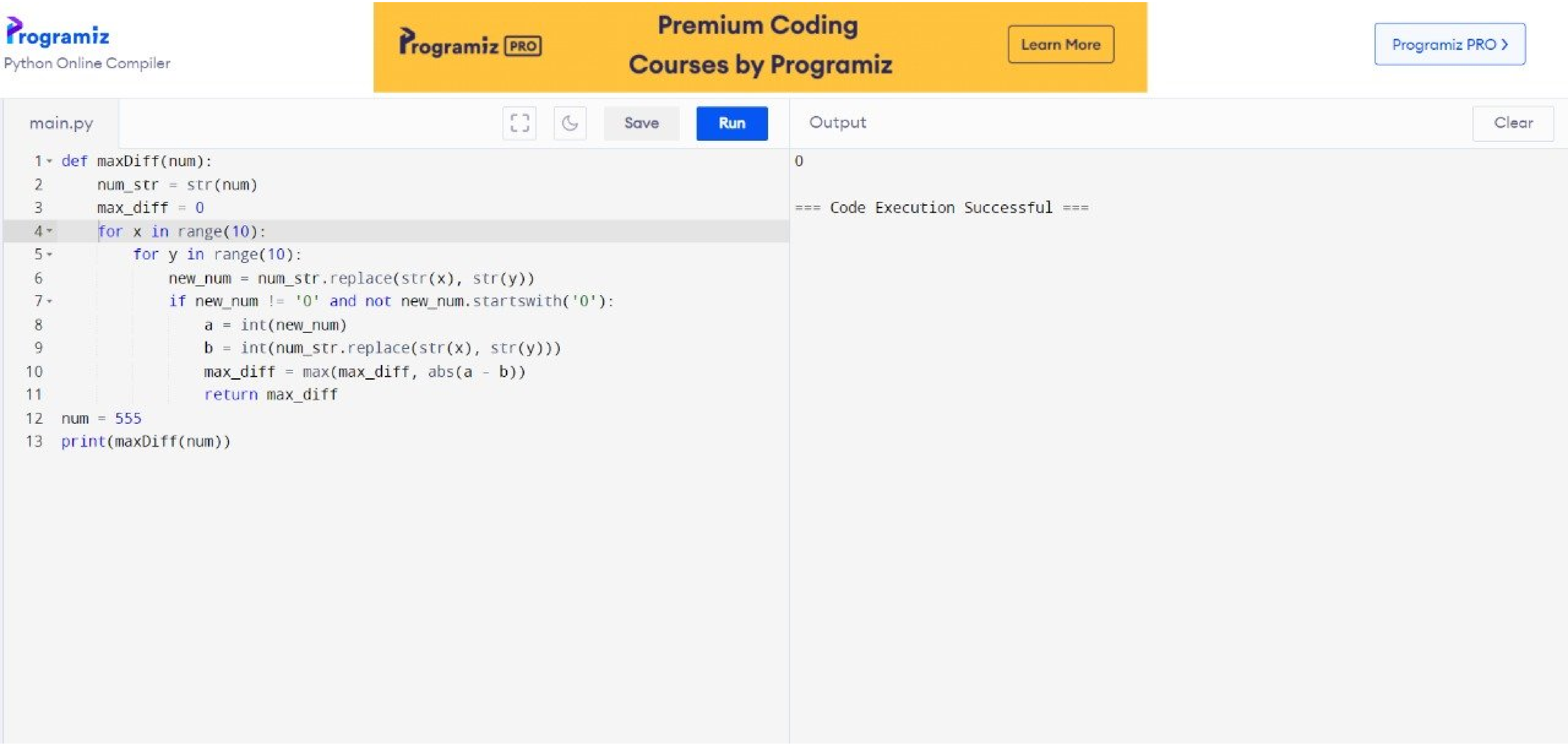
```python
def isValidSequence(root, arr):
    def check(node, i):
        if not node or i == len(arr) or node.val != arr[i]:
            return False
        if not node.left and not node.right:
            return i == len(arr) - 1
        return check(node.left, i + 1) or check(node.right, i + 1)

    return check(root, 0)

```

```
=== Code Execution Successful ===
```



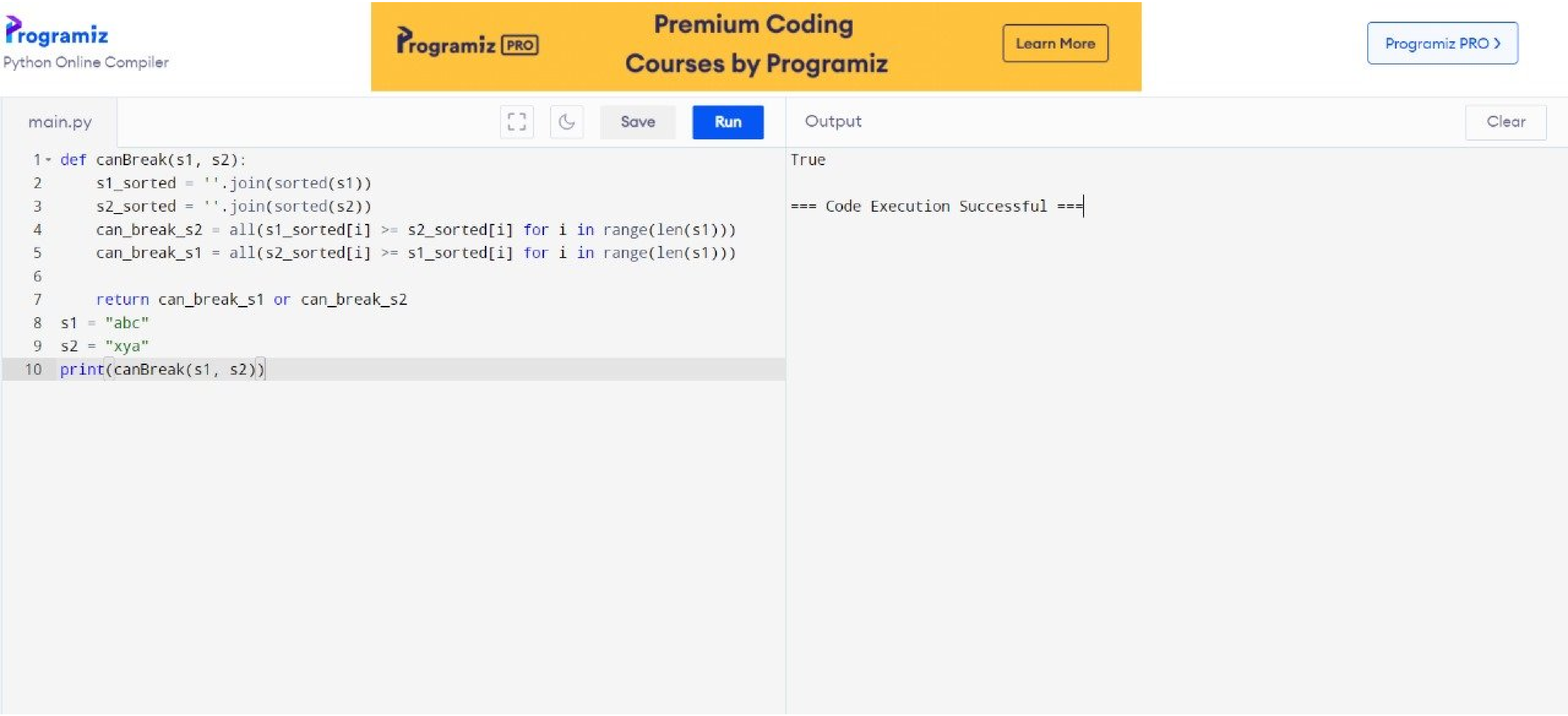
```python
def leftmost_column_with_one(binaryMatrix):
    rows = len(binaryMatrix)
    cols = len(binaryMatrix[0])
    row = 0
    col = cols - 1
    leftmost_col = -1
    while row < rows and col >= 0:
        if binaryMatrix[row][col] == 1:
            leftmost_col = col
            col -= 1
        else:
            row += 1
    return leftmost_col
binaryMatrix = [
    [0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 1, 1, 1]]
print(leftmost_column_with_one(binaryMatrix))
```

```
1

=== Code Execution Successful ===
```



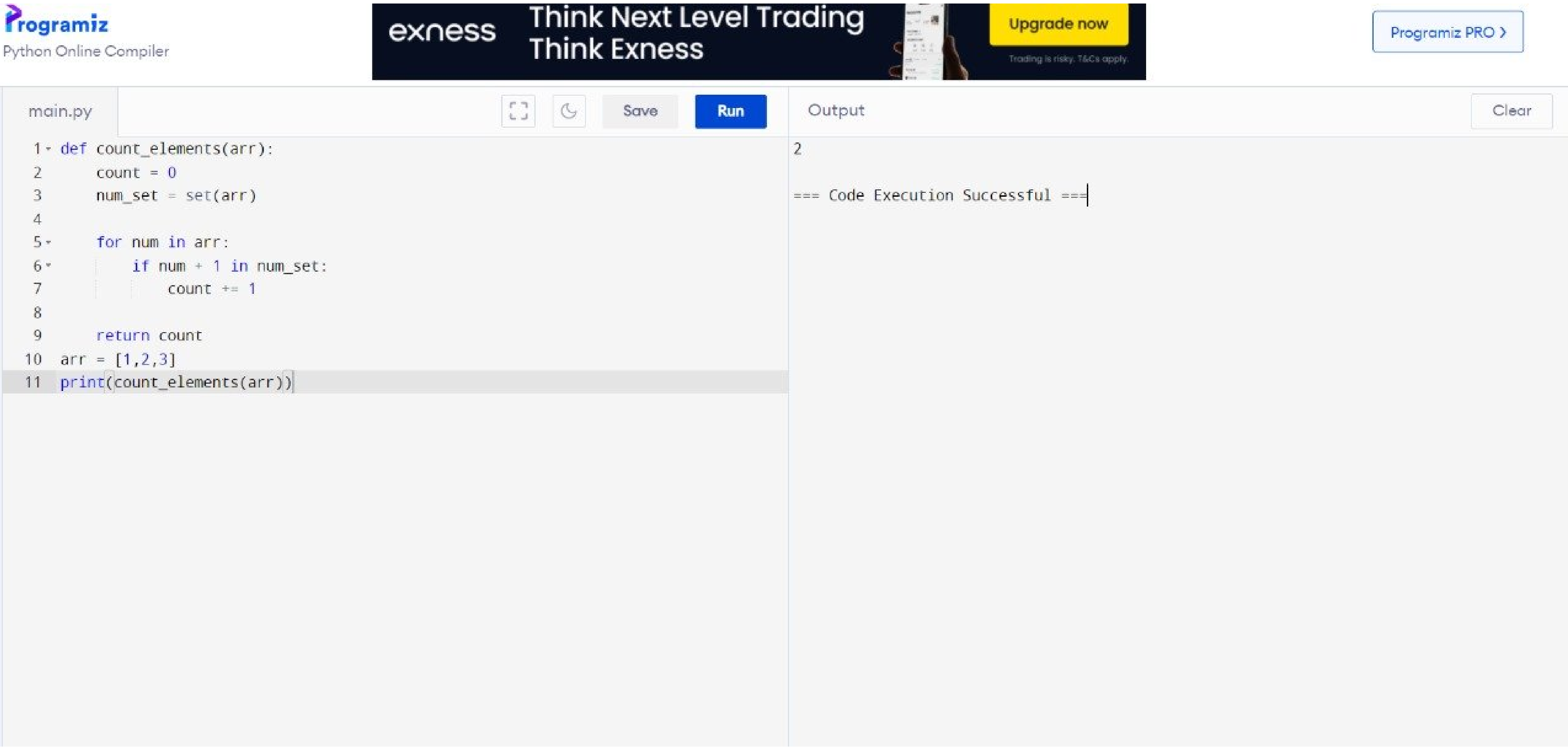
```python
def maxDiff(num):
    num_str = str(num)
    max_diff = 0
    for x in range(10):
        for y in range(10):
            new_num = num_str.replace(str(x), str(y))
            if new_num != '0' and not new_num.startswith('0'):
                a = int(new_num)
                b = int(num_str.replace(str(x), str(y)))
                max_diff = max(max_diff, abs(a - b))
                return max_diff
num = 555
print(maxDiff(num))
```

```
0


=== Code Execution Successful ===
```



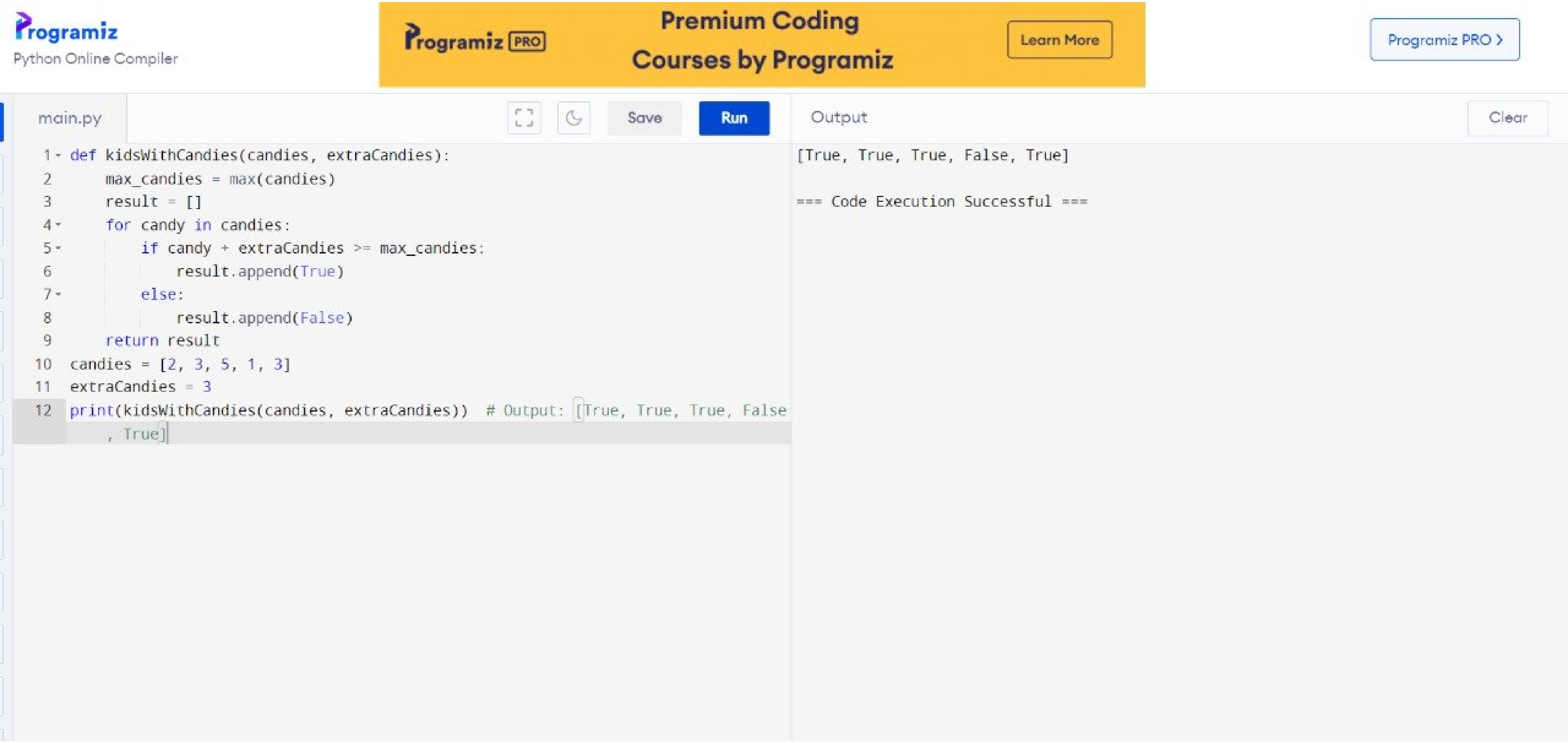
```python
def canBreak(s1, s2):
    s1_sorted = ''.join(sorted(s1))
    s2_sorted = ''.join(sorted(s2))
    can_break_s2 = all(s1_sorted[i] >= s2_sorted[i] for i in range(len(s1)))
    can_break_s1 = all(s2_sorted[i] >= s1_sorted[i] for i in range(len(s1)))

    return can_break_s1 or can_break_s2
s1 = "abc"
s2 = "xya"
print(canBreak(s1, s2))
```

```
True

=== Code Execution Successful ===
```



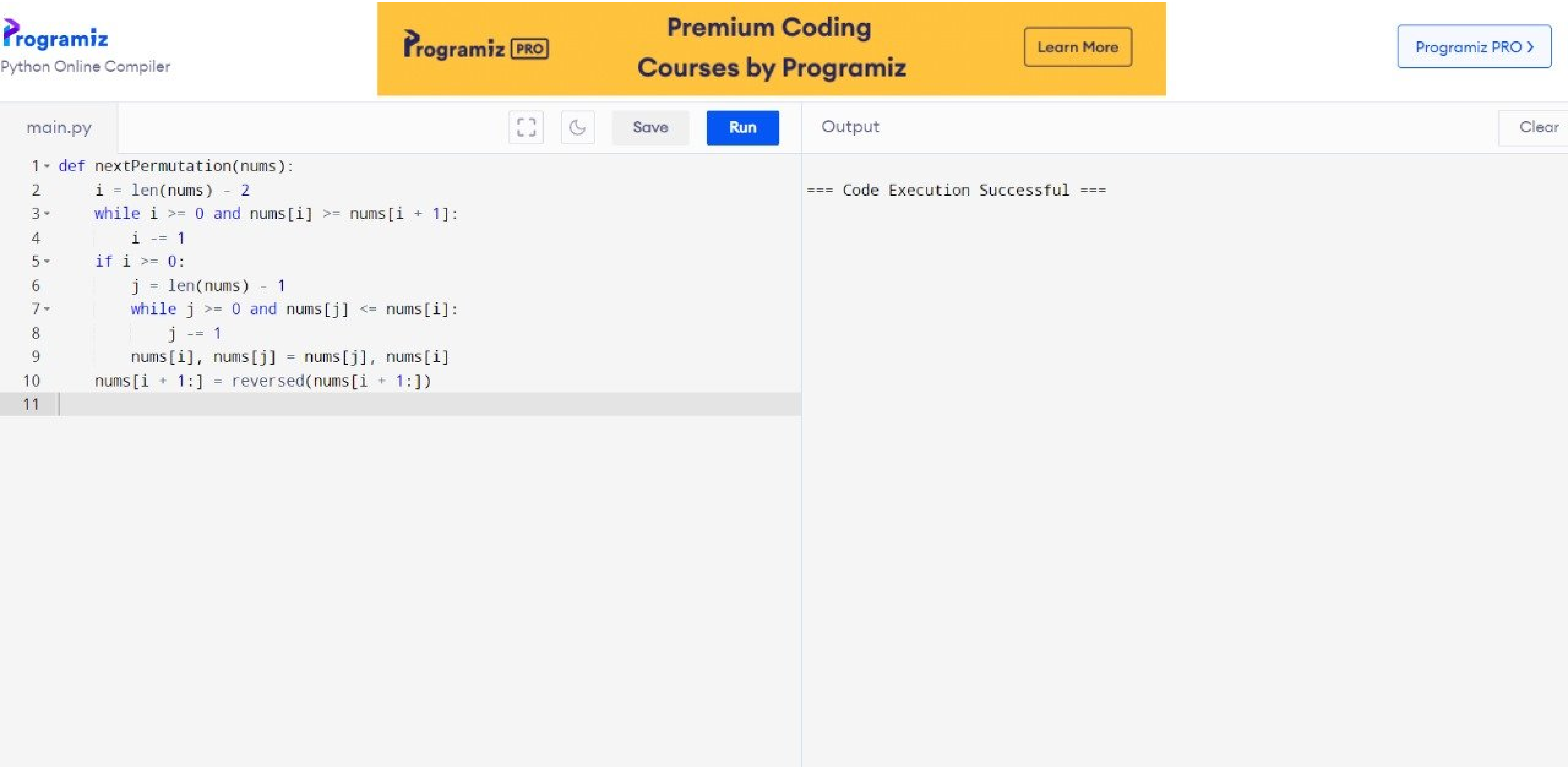
```python
def count_elements(arr):
    count = 0
    num_set = set(arr)

    for num in arr:
        if num + 1 in num_set:
            count += 1

    return count
arr = [1,2,3]
print(count_elements(arr))
```

```
2

=== Code Execution Successful ===
```



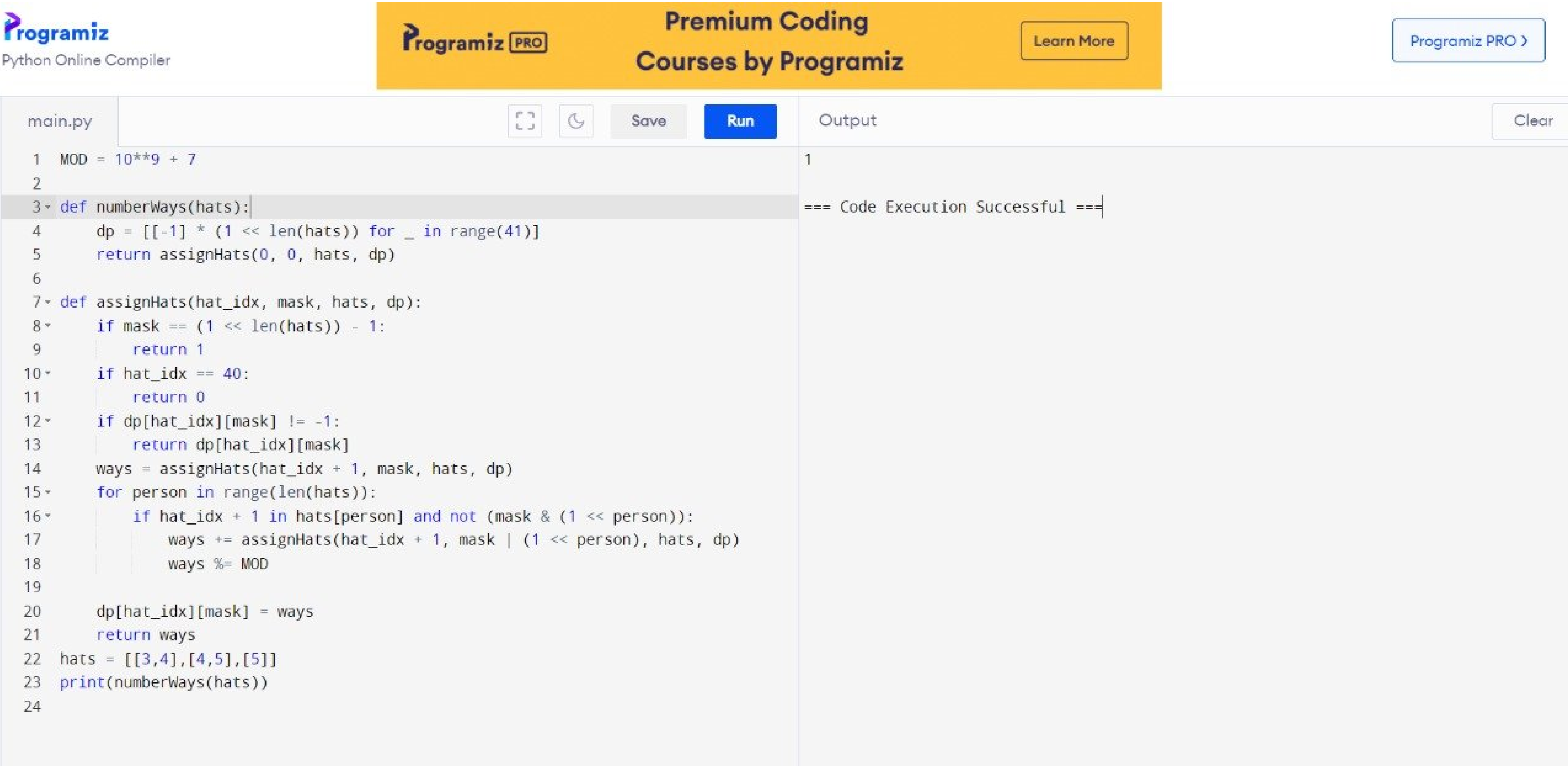
```python
def kidsWithCandies(candies, extraCandies):
    max_candies = max(candies)
    result = []
    for candy in candies:
        if candy + extraCandies >= max_candies:
            result.append(True)
        else:
            result.append(False)
    return result
candies = [2, 3, 5, 1, 3]
extraCandies = 3
print(kidsWithCandies(candies, extraCandies))  # Output: [True, True, True, False, True]
```

```
[True, True, True, False, True]

=== Code Execution Successful ===
```



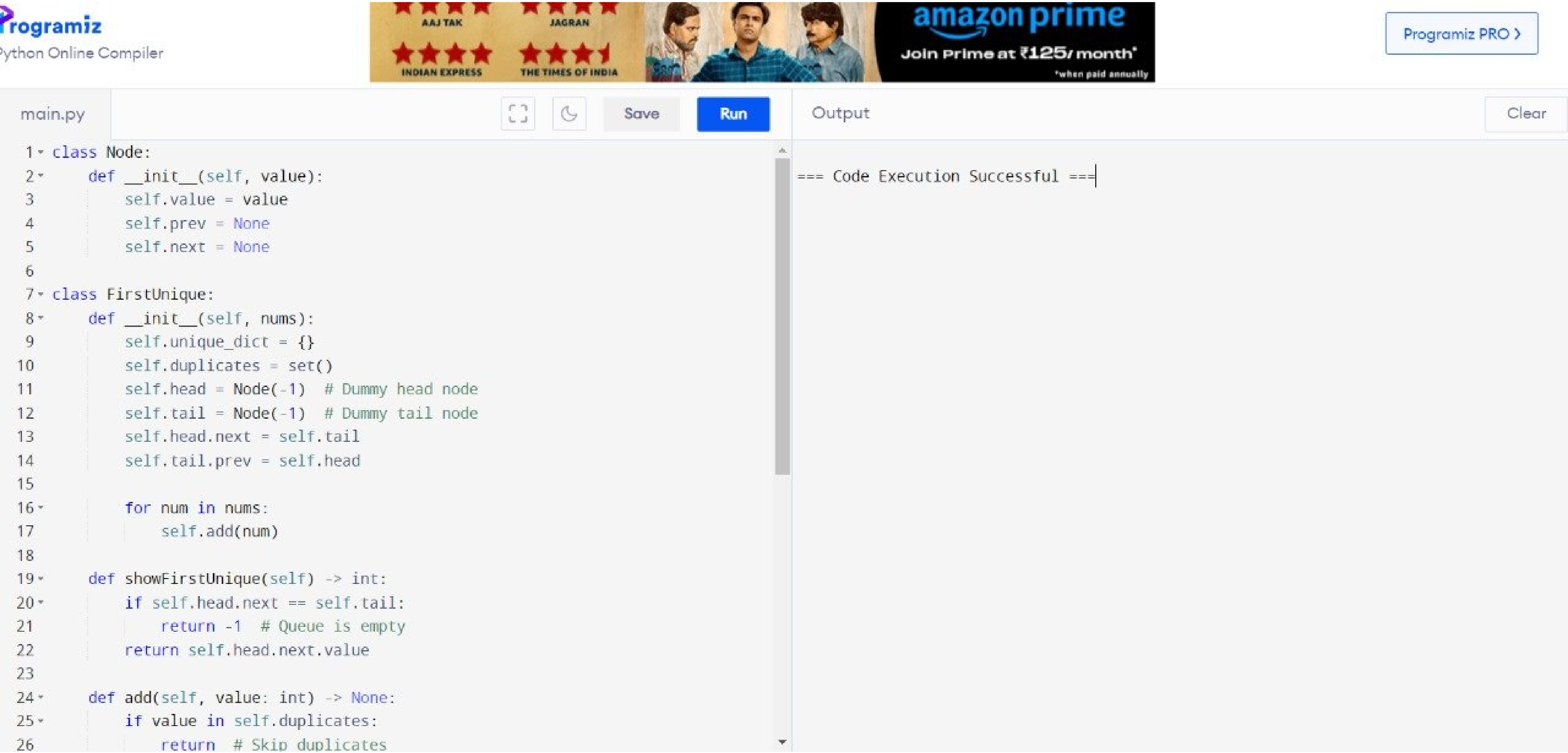
```python
def nextPermutation(nums):
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    if i >= 0:
        j = len(nums) - 1
        while j >= 0 and nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]
    nums[i + 1:] = reversed(nums[i + 1:])

```

```
=== Code Execution Successful ===
```



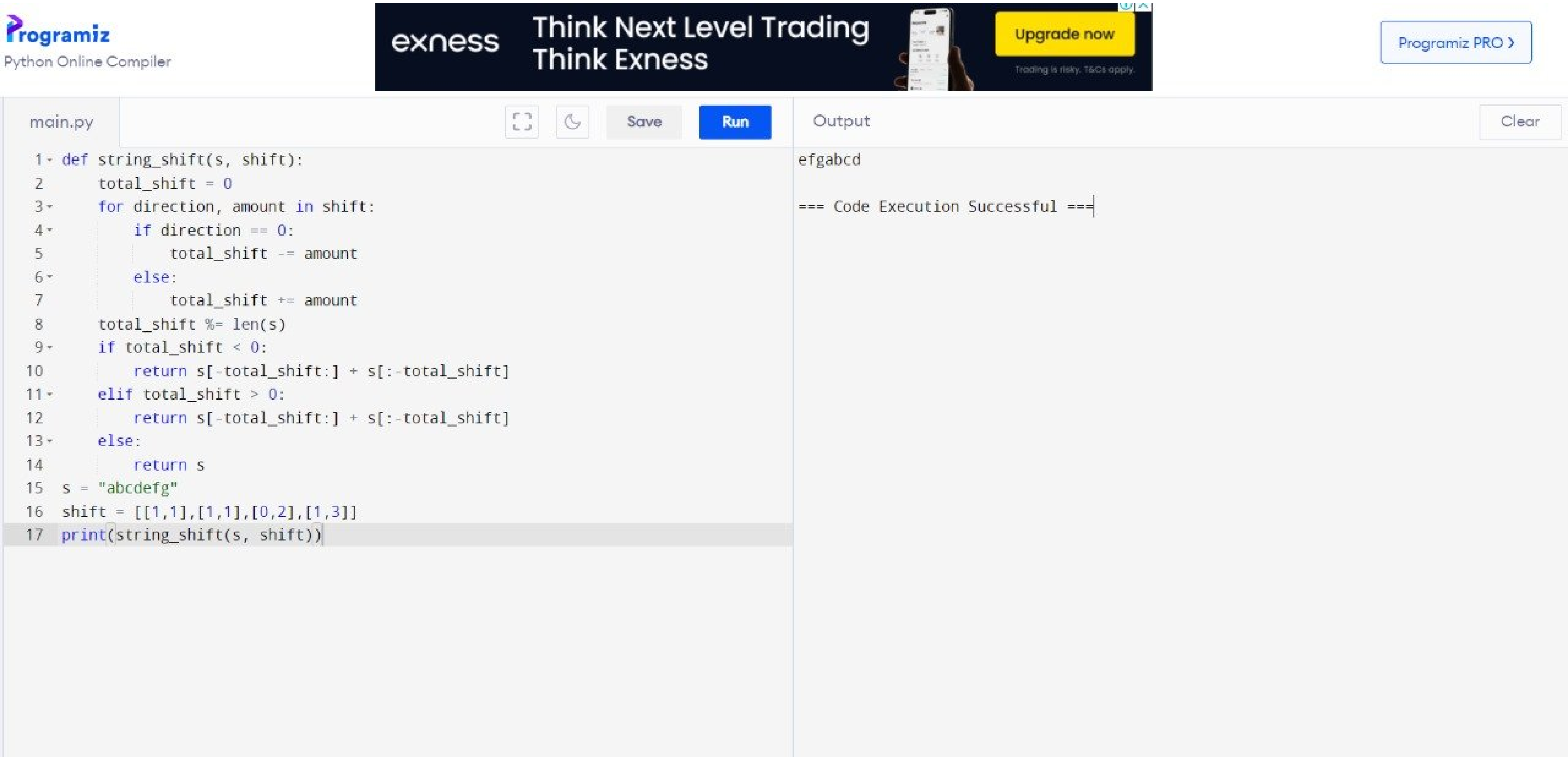
```python
MOD = 10**9 + 7

def numberWays(hats):
    dp = [[-1] * (1 << len(hats)) for _ in range(41)]
    return assignHats(0, 0, hats, dp)

def assignHats(hat_idx, mask, hats, dp):
    if mask == (1 << len(hats)) - 1:
        return 1
    if hat_idx == 40:
        return 0
    if dp[hat_idx][mask] != -1:
        return dp[hat_idx][mask]
    ways = assignHats(hat_idx + 1, mask, hats, dp)
    for person in range(len(hats)):
        if hat_idx + 1 in hats[person] and not (mask & (1 << person)):
            ways += assignHats(hat_idx + 1, mask | (1 << person), hats, dp)
            ways %= MOD

    dp[hat_idx][mask] = ways
    return ways
hats = [[3,4],[4,5],[5]]
print(numberWays(hats))
```

```
1

=== Code Execution Successful ===
```



```
class Node:
    def __init__(self, value):
        self.value = value
        self.prev = None
        self.next = None

class FirstUnique:
    def __init__(self, nums):
        self.unique_dict = {}
        self.duplicates = set()
        self.head = Node(-1)  # Dummy head node
        self.tail = Node(-1)  # Dummy tail node
        self.head.next = self.tail
        self.tail.prev = self.head

        for num in nums:
            self.add(num)

    def showFirstUnique(self) -> int:
        if self.head.next == self.tail:
            return -1  # Queue is empty
        return self.head.next.value

    def add(self, value: int) -> None:
        if value in self.duplicates:
            return  # Skip duplicates
```

```
=== Code Execution Successful ===
```



```python
def string_shift(s, shift):
    total_shift = 0
    for direction, amount in shift:
        if direction == 0:
            total_shift -= amount
        else:
            total_shift += amount
    total_shift %= len(s)
    if total_shift < 0:
        return s[-total_shift:] + s[:-total_shift]
    elif total_shift > 0:
        return s[-total_shift:] + s[:-total_shift]
    else:
        return s
s = "abcdefg"
shift = [[1,1],[1,1],[0,2],[1,3]]
print(string_shift(s, shift))
```

```
efgabcd

=== Code Execution Successful ===
```